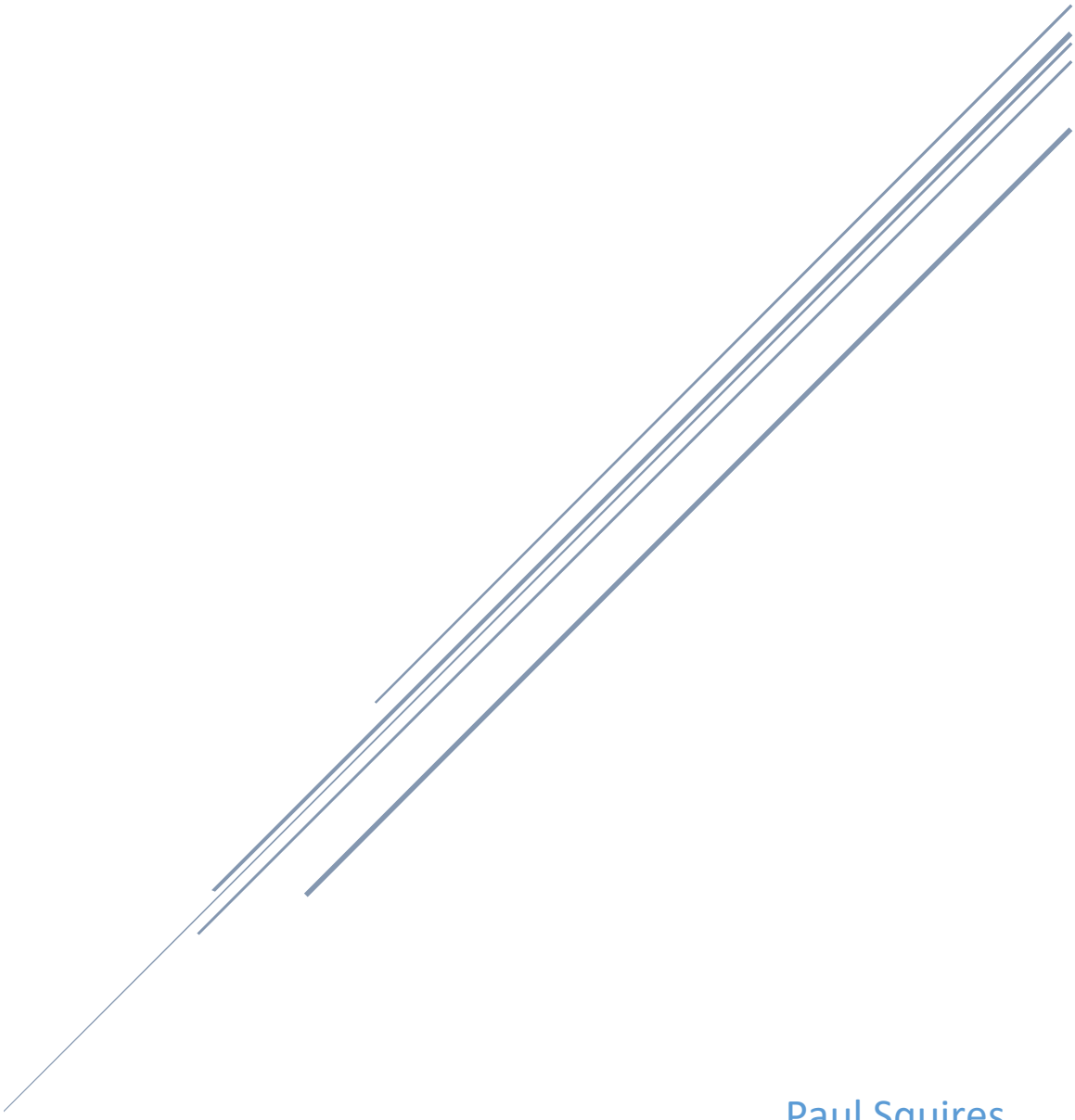
CPRINTPREVIEW

Reference Manual



Paul Squires
planetsquires.com

Introduction

The CPrintPreview class is an attempt to provide a simple and easy way for the programmer to create documents (via code) that can be viewed on screen and/or printed to a printer. It is designed to work with the 32-bit and 64-bit versions of the FreeBasic programming language.

The CPrintPreview class requires the WinFBX library authored by Jose Roca and can be found at https://github.com/JoseRoca/WinFBX If you are a WinFBE Editor Suite user then the WinFBX library is already included with the editor and is already available for your use. When you see variables in this documentation like *CWSTR* then realize that it refers to the variable width Unicode wide string class from the WinFBX library. You can download the WinFBE Editor Suite (which includes WinFBX and also the latest FreeBasic compiler) from https://github.com/PaulSquires/WinFBE

CPrintPreview is inspired in the vein of commercial products like *Virtual Print Engine (VPE)* by IDEAL Software. Although incredibly powerful, *VPE* is a quite expensive product and most likely out of reach for most hobby programmers. The free community version of *VPE* does not provide enough features or flexibility for most use cases and it is not high dpi aware which can be a deal breaker for most programs.

Even though CPrintPreview is small and lightweight, it does provide several great features that rival commercial products, such as:

- All source code. No need for DLL for ActiveX.
- Incredibly fast! You can generate a hundred pages in a fraction of a second.
- Canvas pages are stored in standard memory using EMF (Enhanced Memory Files).
- You can design pages freely allowing you to jump between pages as you require rather than having to start at the first page and create pages sequentially.
- High DPI aware.
- The preview window can be used standalone and not tied to a predefined preview dialog.
- Preview windows can be sent directly to the printer without having to first be shown.
- The user is free to design their own custom print preview dialogs. You are not tied to a predefined dialog that would be difficult to customize.

Of course, CPrintPreview does have some limitations, such as:

- Images and picture support is not available yet.
- Barcode support is not available yet.
- Export to other formats such as PDF is not available yet.
- There is no design time report creation tool. Canvas must be created via code.

Quick Start

It is very quick to get up and running with CPrintPreview.

There are only two source files for the class and they are found in the *src* folder of the download.

CPrintPreview.bi (this is the class header file)
CPrintPreview.inc (this is the actual class code source)

Depending on your programming style, you can either compile CPrintPreview.inc into an object file and link to your main files, or you can simply #Include "CPrintPreview.inc" into your source files and compile it together with all of your other source files.

In today's world, most computers are so fast that simply #Include'ing the source files is the easiest approach.

The *demo* folder located in the download contains a simple framework that shows how to use the class in a real world setting. Granted, the demo is pretty barebones but it does explain how to create the print preview canvas, embed that canvas into a print preview dialog, and then show and/or print that canvas.

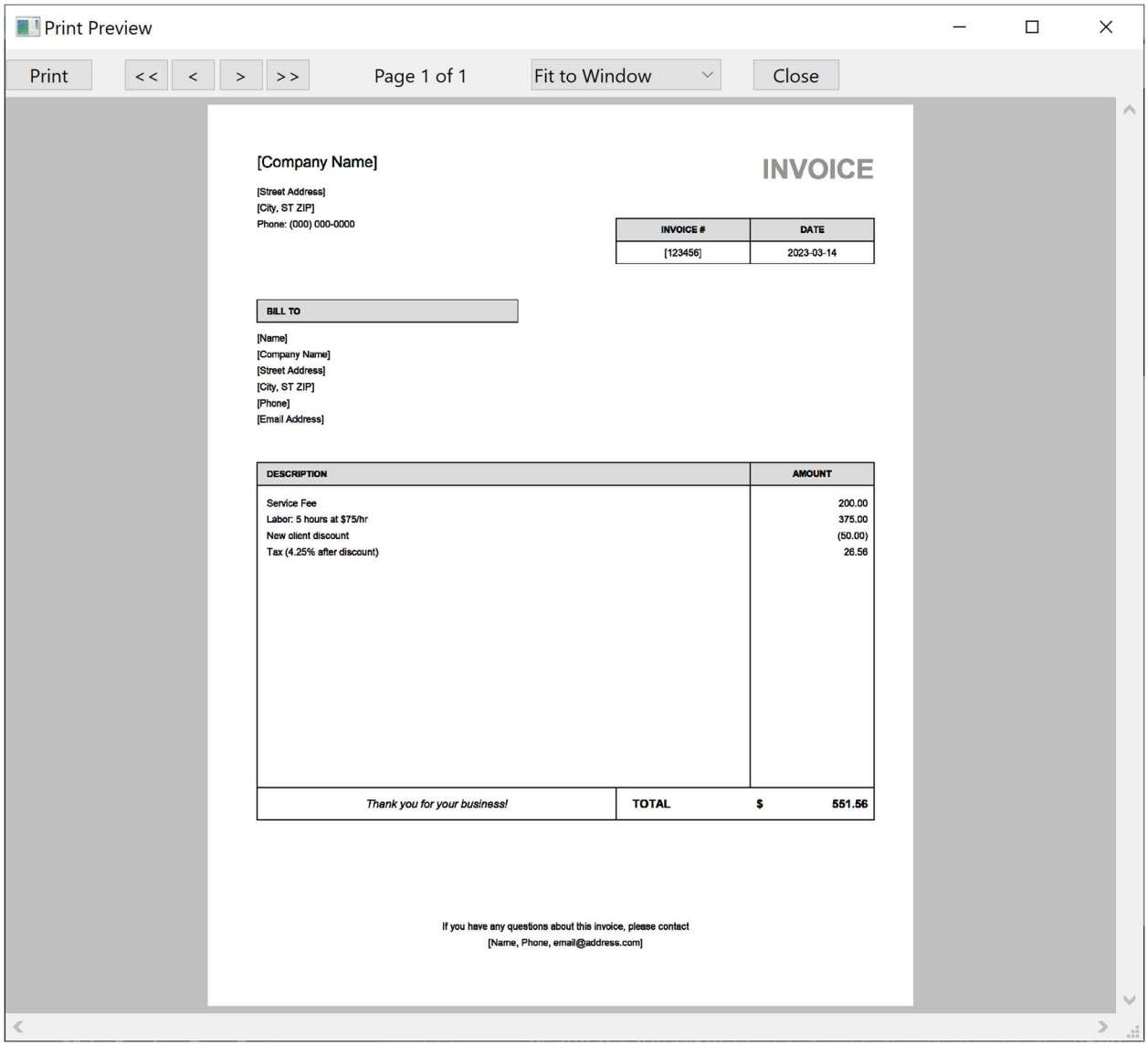
For example, in your main application you could press a button or respond to some other event and initiate the creation of the print preview canvas:

```
' Create an in-memory print preview Canvas
dim pCanvas as CPrintPreview ptr = CreatePrintPreviewCanvas()

' Pass our Canvas to our dialog that displays the print preview. If we
' simply wanted to output the Canvas to the printer then we could
' set the printer properties to the Canvas and call pCanvas->PrintDoc
ShowPrintPreviewDlg( HWND, pCanvas )
```

It is the *`CreatePrintPreviewCanvas()`* function that does all of the heavy work to design the canvas and return a pointer to the canvas window that can then be embedded into a print preview dialog. This is a function that you will write in your own program.

The following is an image of a canvas embedded in a print preview dialog that was created to simulate a real world example of creating an invoice that could be printed or emailed to a customer. Following the image is the code from a *`CreatePrintPreviewCanvas()`* that was used to create the canvas.



Here is the code used to create the above canvas. We'll explain the usage of that code step by step.

```
function CreatePrintPreviewCanvas() as CPrintPreview ptr

  ' Create the new Invoice document

   dim pCanvas as CPrintPreview ptr = new CPrintPreview

   ' Add different fonts to the Canvas that we can use. These fonts are
   ' automatically released when the Canvas is destroyed (to avoid GDI leaks).
   dim as HFONT hFont(5)
   hFont(0) = pCanvas->AddFont( "Arial", 9, true )      ' Invoice/Date headings
   hFont(1) = pCanvas->AddFont( "Arial", 10 )           ' Regular line items
   hFont(2) = pCanvas->AddFont( "Arial", 11, , true )   ' Thank you for your business!
   hFont(3) = pCanvas->AddFont( "Arial", 12, true )     ' Total amount
   hFont(4) = pCanvas->AddFont( "Arial", 16 )           ' [COMPANY NAME]
   hFont(5) = pCanvas->AddFont( "Arial", 28, true )     ' INVOICE (gray color)

   ' Create a solid pen of width 2. Black color is used if no color is specified.
```

```
   dim as HPEN hPen = pCanvas->AddSolidPen( 2 )
   ' Create an invisible pen used for writing text to boxes with no border.
   dim as HPEN hPenInvisible = pCanvas->AddInvisiblePen()

   ' Define gray color for solid boxes
   dim as COLORREF clrLtGray = BGR(219,219,219)
   dim as COLORREF clrDkGray = BGR(148,148,148)

   ' Need to set the measurement units before creating objects if
   ' you decide to use centimeters because inches is the default.
   pCanvas->MeasurementUnits = 1   ' 0:Inches, 1:Centimeters

   pCanvas->Orientation = DMORIENT_PORTRAIT
   pCanvas->PaperWidth = 21.59     ' cm  (8.5 inches)
   pCanvas->PaperHeight = 27.94    ' cm  (11 inches)

   ' Page #0
   pCanvas->ModifyPage(0)

   pCanvas->UseFont( hFont(4) )
   ' black text color and white background is used if not specified
   pCanvas->WriteText( 1.5, 1.5, "[Company Name]" )

   pCanvas->UseFont( hFont(1) )
   pCanvas->WriteText( 1.5, 2.5, "[Street Address]" )
   pCanvas->WriteText( 1.5, 3.0, "[City, ST ZIP]" )
   pCanvas->WriteText( 1.5, 3.5, "Phone: (000) 000-0000" )

   pCanvas->UseFont( hFont(5) )
   pCanvas->WriteTextBox( 16.0, 1.5, 20.4, 2.5, "INVOICE", hPenInvisible, , DT_TOP or DT_RIGHT,
clrDkGray )

   pCanvas->UseFont( hFont(0) )
   pCanvas->WriteTextBox( 12.5, 3.5, 16.6, 4.2, "INVOICE #", hPen, clrLtGray )
   pCanvas->WriteTextBox( 16.6, 3.5, 20.4, 4.2, "DATE", hPen, clrLtGray )
   pCanvas->UseFont( hFont(1) )
   pCanvas->WriteTextBox( 12.5, 4.2, 16.6, 4.9, "[123456]", hPen )
   pCanvas->WriteTextBox( 16.6, 4.2, 20.4, 4.9, "2023-03-14", hPen )

   pCanvas->UseFont( hFont(0) )
   pCanvas->DrawSolidRect( 1.5, 6.0, 9.5, 6.7, hPen, clrLtGray )
   pCanvas->WriteText( 1.8, 6.2, "BILL TO", , clrLtGray )
   pCanvas->UseFont( hFont(1) )
   pCanvas->WriteText( 1.5, 7.0, "[Name]" )
   pCanvas->WriteText( 1.5, 7.5, "[Company Name]")
   pCanvas->WriteText( 1.5, 8.0, "[Street Address]" )
   pCanvas->WriteText( 1.5, 8.5, "[City, ST ZIP]" )
   pCanvas->WriteText( 1.5, 9.0, "[Phone]" )
   pCanvas->WriteText( 1.5, 9.5, "[Email Address]" )

   pCanvas->UseFont( hFont(0) )
   pCanvas->DrawSolidRect( 1.5, 11.0, 16.6, 11.7, hPen, clrLtGray )
   pCanvas->WriteText( 1.8, 11.2, "DESCRIPTION", , clrLtGray )
   pCanvas->WriteTextBox( 16.6, 11.0, 20.4, 11.7, "AMOUNT", hPen, clrLtGray )
   pCanvas->DrawRect( 1.5, 11.7, 16.6, 21.0, hPen )
   pCanvas->DrawRect( 16.6, 11.7, 20.4, 21.0, hPen )

   pCanvas->UseFont( hFont(1) )
   pCanvas->WriteTextBox( 1.8, 12.0, 16.0, 12.5, "Service Fee", hPenInvisible, , DT_LEFT or
DT_VCENTER )
   pCanvas->WriteTextBox( 1.8, 12.5, 16.0, 13.0, "Labor: 5 hours at $75/hr", hPenInvisible, ,
DT_LEFT or DT_VCENTER )
   pCanvas->WriteTextBox( 1.8, 13.0, 16.0, 13.5, "New client discount", hPenInvisible, , DT_LEFT
or DT_VCENTER )
   pCanvas->WriteTextBox( 1.8, 13.5, 16.0, 14.0, "Tax (4.25% after discount)", hPenInvisible, ,
DT_LEFT or DT_VCENTER )

   pCanvas->WriteTextBox( 18.2, 12.0, 20.2, 12.5, "200.00", hPenInvisible, , DT_RIGHT or
DT_VCENTER )
   pCanvas->WriteTextBox( 18.2, 12.5, 20.2, 13.0, "375.00", hPenInvisible, , DT_RIGHT or
DT_VCENTER )
```

```
   pCanvas->WriteTextBox( 18.2, 13.0, 20.2, 13.5, "(50.00)", hPenInvisible, , DT_RIGHT or
DT_VCENTER )
   pCanvas->WriteTextBox( 18.2, 13.5, 20.2, 14.0, "26.56", hPenInvisible, , DT_RIGHT or
DT_VCENTER )

   pCanvas->UseFont( hFont(2) )
   pCanvas->WriteTextBox( 1.5, 21.0, 12.5, 22.0, "Thank you for your business!", hPen )

   pCanvas->UseFont( hFont(3) )
   pCanvas->DrawRect( 12.5, 21.0, 20.4, 22.0, hPen )
   pCanvas->WriteTextBox( 13.0, 21.2, 16.6, 21.8, "TOTAL", hPenInvisible, , DT_LEFT or DT_VCENTER
)
   pCanvas->WriteTextBox( 16.8, 21.2, 18, 21.8, "$", hPenInvisible, , DT_LEFT or DT_VCENTER  )
   pCanvas->WriteTextBox( 18.2, 21.2, 20.2, 21.8, "551.56", hPenInvisible, , DT_RIGHT or
DT_VCENTER )

   dim as CWSTR wszText
   pCanvas->UseFont( hFont(1) )
   wszText = "If you have any questions about this invoice, please contact"
   pCanvas->WriteTextBox( 1.5, 25.0, 20.4, 25.5, wszText, hPenInvisible )
   wszText = "[Name, Phone, email@address.com]"
   pCanvas->WriteTextBox( 1.5, 25.5, 20.4, 26.0, wszText, hPenInvisible )

   function = pCanvas
end function
```

As you can see from the above code, it is very easy and intuitive to create even somewhat complex canvas documents.

Here is the normal sequence of code that you would usually follow:

Create the canvas pointer.

```
dim pCanvas as CPrintPreview ptr = new CPrintPreview
```

This is a pointer to a *new* memory allocated CPrintPreview class and it is this pointer that is returned to the calling program. The calling program can then do whatever it wishes with the newly created class such as embed the canvas window (*AttachToParentWindow*) into a print preview dialog for the user to interact with, or simply print the canvas to a printer (*PrintDoc*).

It is the calling program's responsibility to *delete* the canvas pointer so that there are no memory leaks resulting from an undestroyed pointer. Destroying the canvas pointer also automatically calls the canvas's destructor so any fonts or pens (GDI objects) that were created during the creation of the canvas will also be destroyed. A good practice is to check the *IsPrinting* method to ensure that it is safe to delete the pointer.

```
' Delete the pCanvas pointer
if pCanvas then
   ' Do not delete the Canvas until it is safe to do so. If we delete
   ' the Canvas while a job is processing to the print queue then we
   ' could get a GPF.
   do until pCanvas->IsPrinting = false
   loop
   Delete(pCanvas)
end if
```

Following the creation of the canvas pointer, you would normally create a series of font, pen and colors that you would employ in the creation of your document. The syntax for those function calls are found

elsewhere in this document. In normal programming, the creation of fonts and pens would entail that the programmer is responsible for "cleaning up" or deleting these objects once they are no longer needed otherwise a GDI memory leak would occur. For convenience, the class tracks these object handles for you and will destroy them automatically once the class itself is destroyed. If you have a need to destroy the objects manually then you can call *DisposeFonts* or *DisposePens* to destroy them.

Most people know what a font is but a "pen" may be a foreign concept. A pen is simply a graphical object that is used to draw lines, curves and outlines (borders) of rectangles and shapes. The pen has a style such as solid, dashed, dotted, invisible; a width; and a color. Refer to the *AddPen* reference section later in this document. An "invisible" pen is simply a pen that can be used in the drawing of rectangles without a visible border.

The next section of code and steps are important to setup the canvas in order to facilitate the subsequent drawing of text, boxes, etc on the canvas. The canvas must know the unit of measurements being used (inches or centimeters), the orientation of the canvas (landscape or portrait) and the height and width of the canvas (basically the paper size being used as if it was in portrait mode).

Following the canvas setup, you will start to modify pages on the canvas via the *ModifyPage* function. Page numbers start at number zero (0) rather than one (1) and you can create and work on your pages in any order that you wish as long as you call the *ModifyPage* function using the desired page number.

Now you are ready to actually create the visual elements of the page.

You will use a combination of functions to tell the canvas what font to use, where to write text, where to draw text enclosed in transparent or shaded boxes/rectangles. All of these (and more) drawing functions are explained in the sections that follow in this document.

Finally, you return the canvas pointer to the calling application.

Function Reference

AddFont

Creates and adds a new font to the canvas. The font handle of the created font is returned. The programmer will subsequently use this font via the *UseFont* function to specify the current font to be used for outputting text to the canvas via the *WriteText* or *WriteTextBox* functions.

The programmer does not need to destroy font handles because the class will automatically destroy all font handles when the class itself is destroyed. However, should you have a specific need to do so, you can manually destroy all font handles via a call to *DisposeFonts*.

Parameters:

FontName as wstring
FontSize as long
FontBold as boolean = false
FontItalic as boolean = false
FontUnderline as boolean = false
FontStrikeout as boolean = false

Some parameters have default values that can be omitted when the function is called.

Example:
Dim as HFONT hMyFont = pCanvas->AddFont(“Arial”, 12, true, false, true, false)
Dim as HFONT hMyFont = pCanvas->AddFont(“Arial”, 12)

AddPen

Creates and adds a new pen to the canvas. The pen handle of the created font is returned. The programmer will subsequently pass this pen handle to various class functions that require it such as the *DrawLine*, *DrawRect*, *DrawSolidRect*, and *WriteTextBox* functions.

The programmer does not need to destroy pen handles because the class will automatically destroy all pen handles when the class itself is destroyed. However, should you have a specific need to do so, you can manually destroy all pen handles via a call to *DisposePens*.

Parameters:

Style as long
Width as long
PenColor as COLORREF = 0

Some parameters have default values that can be omitted when the function is called.

Style can be any of the following constants. You can use and of the constants found in Microsoft’s documentation for the Win32 api CreatePen function but these are the most commonly used:

PS_SOLID The pen is solid.

PS_DASH	The pen is dashed. Pen width must be one or less.
PS_DOT	The pen is dotted. Pen width must be one or less.
PS_DASHDOT	The pen has alternating dashes and dots. Pen width must be one or less.
PS_DASHDOTDOT	The pen has alternating dashes and double dots. Pen width must be one or less.
PS_NULL	The pen is invisible.

The default pen color is black but you can set it to whatever color you wish by passing in an RGB color value. In FreeBasic, you use the **BGR** macro (not RGB), in order to specify the Red, Green, Blue values.

Also, refer to several other more specific pen creation functions that may make your code more self documenting. *AddSolidPen*, *AddDashPen*, *AddDotPen*, *AddInvisiblePen*

```
AddSolidPen(Width as long, clr as COLORREF = 0 )
AddDashPen( clr as COLORREF = 0 )
AddDotPen( clr as COLORREF = 0 )
AddInvisiblePen()
```

An “invisible” pen is simply a pen that does not draw anything but is necessary to be passed to certain drawing functions that require a pen. For example, you may wish to output text to a box/rectangle but not want that box/rectangle to actually have a drawn border. In this case, you would pass an invisible pen.

Example:

```
‘ Create a solid red pen of 4 units wide
Dim as HPEN hMyRedPen = pCanvas->AddPen( PS_SOLID, 4, BGR(255,0,0) )
```

AttachToParentWindow

You do not have to “print preview” your created canvas. You have the option to simply output your canvas directly to a printer (see *PrintDoc*). In cases where you want your user to see the canvas, you would embed the canvas in a parent window. That parent window will most likely be in some sort of custom print preview dialog that you design. The *demo* that comes in the CPrintPreview download has a sample print preview dialog.

By calling *AttachToParentWindow*, the class creates a standard Windows control as a child of the parent window that you specify. You also pass to the function a control ID identifier so that you can easily refer to your canvas window. When the canvas is attached, a WM_SIZE message is sent your parent window so that you can easily position and size the canvas within your dialog.

Parameters:

```
hWndParent as HWND
ControlId as long
```

Example:

```
const IDC_CANVAS = 1000
pCanvas->AttachToParentWindow( hDlg, IDC_CANVAS )
```

ChoosePrinter

Displays the standard Windows popup *ChoosePrinter* dialog where you can select and configure the printer of your choice.

Parameters:
hWndParent as HWND

Example:
pCanvas->ChoosePrinter(hDlg)

Copies

Sets or returns the number of copies to print.

Parameters:
NumCopies as long

Example:
pCanvas->Copies = 5
nCopies = pCanvas->Copies

CornerRadius

Sets or returns the curvature radius to use for rounded corners of boxes/rectangles.

Parameters:
CornerRadius as long

Example:
pCanvas->CornerRadius = 80
nRadius = pCanvas->CornerRadius

CurrentPage

Sets or returns the current (zero based) page. This is useful when you wish to set what visual page will first show to the user when the print preview is displayed. If you wish to create or change the contents of a page then use the *ModifyPage* function instead.

Parameters:
PageNum as long

Example:
pCanvas->CurrentPage = 2
nPage = pCanvas->CurrentPage

DefaultPrinterName

Returns the name of the current default printer as defined in the user's Windows configuration. This is also the printer that is automatically used by default when your canvas is first created. To change to a different printer you can select one from the *ChoosePrinter* popup dialog, or via the *PrinterName* property.

Example:
dim MyPrinter as CWSTR
MyPrinter = pCanvas->DefaultPrinterName

DisposeFonts

The canvas control will automatically destroy and free memory for any fonts that you add to the canvas via the *AddFont* function. This occurs when the CPrintPreview class itself is destroyed. If you have a need to free the fonts memory prior to the destruction of the class then you can call *DisposeFonts*.

Example:
pCanvas->DisposeFonts

DisposePens

The canvas control will automatically destroy and free memory for any pens that you add to the canvas via the add pen functions. This occurs when the CPrintPreview class itself is destroyed. If you have a need to free the pens memory prior to the destruction of the class then you can call *DisposePens*.

Example:
pCanvas->DisposePens

DrawRect

Draws a non-filled (transparent) rectangle based on the supplied coordinates and pen. The coordinates supplied should be in either inches or centimeters depending on the value specified via the *MeasurementUnits* function. The pen will determine the style, width and color of the rectangle's border. You should use an invisible pen (*AddInvisiblePen*) if you do not want any border.

Parameters:
Left as single
Top as single
Right as single
Bottom as single
hPen as HPEN

Example:
pCanvas->DrawRect(1, 1, 5, 5, hBluePen)

DrawSolidRect

Draws a solid filled rectangle based on the supplied coordinates, pen and fill color. The coordinates supplied should be in either inches or centimeters depending on the value specified via the *MeasurementUnits* function. The pen will determine the style, width and color of the rectangle's border. You should use an invisible pen (*AddInvisiblePen*) if you do not want any border.

Parameters:
- Left as single
- Top as single
- Right as single
- Bottom as single
- hPen as HPEN
- FillColor as COLORREF

Example:
```
pCanvas->DrawSolidRect( 1.5, 1, 7.5, 5.25, hRedPen, BGR(0,255,0 )
```

DrawLine

Draws a line based on the supplied coordinates and pen. The coordinates supplied should be in either inches or centimeters depending on the value specified via the *MeasurementUnits* function. The pen will determine the style, width and color of the line.

Parameters:
- Left as single
- Top as single
- Right as single
- Bottom as single
- hPen as HPEN

Example:
```
pCanvas->DrawLine ( 1, 1, 7.5, 1, hPen )
```

IsPrinting

Prior to deleting your canvas pointer, you should check to ensure that the class is not currently processing and output to a printer. The time it takes the class to process and send information to the printer is extremely fast but should you delete the canvas pointer prior to this process completing, then your application may corrupt and GPF.

Example:
```
if pCanvas then
    do until pCanvas->IsPrinting = false
    loop
    Delete(pCanvas)
end if
```

JobName

Sets or returns the print job name. This is the text that identifies the print job that is sent to the printer queue.

Parameters:
JobName as CWSTR

Example:
```
pCanvas->JobName = “My print job”
dim wszName as CWSTR
wszName = pCanvas->JobName
```

MeasurementUnits

Sets or returns the measurement units to be used by the canvas when calculating drawing coordinates and page sizes. You can specify either inches (the default) or centimeters. This setting should be made prior to creating any objects on the canvas.

Parameters:
Units as long ‘ 0:inches, 1:centimeters

Example:
```
pCanvas->MeasurementUnits = 1
units = pCanvas->MeasurementUnits
```

ModifyPage

Sets the page that subsequent drawing commands will act on. The page number is zero based which means that the first page is page zero (0). You can modify pages in any order as long as you call *ModifyPage* prior to doing the modifications.

Parameters:
PageNumber as long

Example:
```
pCanvas->ModifyPage(0)      ‘ modify the first page
```

Orientation

Sets or returns the page orientation to either portrait (the default), or landscape.

Parameters:
Orientation as long ‘ 1:portrait, 2:landscape

Alternatively, you can also use the following two constants if you wish:

DMORIENT_PORTRAIT
DMORIENT_LANDSCAPE

Example:
pCanvas-> Orientation = 1
orientation = pCanvas->Orientation

PageCount

Returns the total number of pages in the canvas that been created.

Example:
numPages = pCanvas->PageCount

PaperHeight

Sets or returns the page height based on the current setting for *MeasurementUnits*. The paper height should always be specified as if the paper is in portrait mode.

Parameters:
PaperHeight as single

Example:
pCanvas->PaperHeight = 27.94 ' cm (11 inches)
paperHeight = pCanvas->PaperHeight

PaperWidth

Sets or returns the page width based on the current setting for *MeasurementUnits*. The paper width should always be specified as if the paper is in portrait mode.

Parameters:
PaperWidth as single

Example:
pCanvas->PaperWidth = 21.59 ' cm (8.5 inches)
paperWidth = pCanvas->PaperWidth

PrintDoc

Sends the canvas to the currently defined printer to be printed.

Example:
pCanvas->PrintDoc

PrinterName

Sets or returns the printer to be attached to the canvas and used for printer output.

Parameters:
PrinterName as CWSTR

Example:
pCanvas->PrinterName = “Microsoft Print to PDF”
dim wszPrinterName as CWSTR
wszPrinterName = pCanvas->PrinterName

UseFont

Sets the font to use for subsequent text output via *WriteText* or *WriteTextBox*. The specified font handle would have previously been created and added to the canvas via *AddFont*.

Parameters:
hFontHandle as HFONT

Example:
pCanvas->UseFont(hFontBold)

VersionNumber

Returns the version of the CPrintPreview class.

Example:
Dim wszVersion as CWSTR
wszVersion = pCanvas->VersionNumber

WriteText

Outputs text at the specified left and top drawing coordinates. The coordinates supplied should be in either inches or centimeters depending on the value specified via the *MeasurementUnits* function.

Parameters:
Left as single
Top as single
Text as CWSTR
TextColor as COLORREF = 0
TextColorBack as COLORREF = -1

If TextColor is not specified then the color will be black

If TextColorBack is not specified then the color will be white.

Example:
pCanvas->WriteText (1, 1, "Here is some red text", BGR(255,0,0))

WriteTextBox

Outputs text within the rectangle region as defined by the supplied drawing coordinates. The coordinates supplied should be in either inches or centimeters depending on the value specified via the *MeasurementUnits* function. The pen will determine the style, width and color of the line. The text output is not clipped to the width of the rectangle.

Parameters:

Left as single
Top as single
Right as single
Bottom as single
Text as CWSTR
hPen as HPEN
FillColor as COLORREF = -1
Alignment as long = DT_CENTER or DT_VCENTER
TextColor as COLORREF = 0
TextColorBack as COLORREF = -1

If hPen is an “invisible” pen then no border will be drawn.

If FillColor is not specified then a hollow brush is used to paint the rectangle background effectively making it transparent.

If TextColor is not specified then the color will be black

If TextColorBack is not specified then the color will be the same as the FillColor (or white if the FillColor was not specified).

Alignment values (values can be combined with each other using the OR keyword):
DT_LEFT, DT_TOP, DT_RIGHT, DT_BOTTOM, DT_CENTER, DT_VCENTER

Example:
dim as COLORREF clrDkGray = BGR(148,148,148)
pCanvas->WriteTextBox(16.0, 1.5, 20.4, 2.5, "INVOICE", hPenInvisible, , DT_RIGHT, clrDkGray)

WindowHandle

Returns the Window handle of the canvas when it has already been attached to a parent window via the *AttachToParentWindow* function.

Example:
Dim hWindow as HWND
hWindow = pCanvas->WindowHandle

Zoom

Sets or returns the level of zoom for the canvas when it is displayed to the user. This setting is only visual and does not affect any printed page.

Parameters:
level as ZoomLevel

ZoomLevel is an enum defined in the class header file.

```
enum ZoomLevel
  FitToWindow = 0
  FitToWidth
  Percent125
  Percent100
  Percent75
end enum
```

Example:
pCanvas->ZoomLevel = ZoomLevel.FitToWindow
level = pCanvas->ZoomLevel